REMARKS

Claims 1, 2, 5, 7, 8, 11, and 13-23 are currently pending. Claim 7 has been amended in accordance with the Patent Office's suggestion. It is respectfully submitted that no new matter has been added.

The Patent Office objected to claim 7 because the word "key" appeared to be missing. Claim 7 has been amended accordingly. It is respectfully submitted that no new matter has been added.

The Patent Office rejected claims 1, 2, 5, 7, 8, 11, and 13-23 under 35 U.S.C. 103(a) as being unpatentable over Shear, U.S. Patent No. 6,157,721, in view of Bodrov, U.S. Patent No. 6,802,006.

As McManis is recited on page 4, lines 10-14, of the Office Action dated November 9, 2006, it is not clear if the Patent Office meant to reject the pending claims over Shear in view of Bodrov and McManis.

Applicant has disclosed the following in the background of the invention:

> In the field of this invention it is known that `Java 2` includes a significantly enhanced security model, compared to previous Java Virtual Machines (JVMs). This new model can restrict the behaviour of a Java applet or application to a clearly defined set of safe actions. This allows a user to download application code from the internet or across a computer network, and run the code in a previously installed JVM. The user can be confident that the application will be assigned the required privileges to function correctly, but to neither damage the user's machine nor divulge sensitive information held on that machine to others via the network or internet. **However, a problem with this approach is that the JVM itself must retain its security integrity in order to ensure downloaded code is restricted in this way. If a malicious user (hacker) has been able to gain access to the user's machine outside of the JVM environment and alter the behaviour of the JVM the whole Java security model is undermined.** For example, the hacker could alter the privileges assigned for software code from a specific source, thereby allowing subsequently downloaded code from this source to function beyond the limits otherwise set by the JVM, and such enhanced privileges could easily be configured to compromise the security integrity of the user's machine. Similarly, the hacker could disable the security code altogether, or worst still insert destructive routines into the core of the JVM which could be activated by an external trigger, such as specific time/date, or when other (possibly

harmless) code is being executed. It is clear that with this malicious activity, early detection of such a compromise of the JVM core would be very useful, and could prevent more serious subsequent damage. If a malicious user decides to attack a machine, the JVM is an obvious target due to its significance in relation to web-based applications, servers and the like. Therefore the security integrity of the JVM is a highly significant factor in the security of the computer as a whole. **A need therefore exists for a software verification system, method and computer program element wherein the abovementioned disadvantages may be alleviated.**

Claim 1 recites

A verification system for a computer software installation, comprising a primary library file, the primary library file having a digital signature; a loader program that obtains a digital signature key and further loads the primary library file, wherein, if a public key cannot be obtained via a virtual machine provider, the digital signature key is a hidden public key internal to the loader program and, if a public key can be obtained via the virtual machine provider, the digital signature key is the public key obtained via the virtual machine provider; and a plurality of secondary files referenced by the primary library file, each of the plurality of secondary files having a digital signature; wherein **the loader program verifies and selectively loads the primary library file** by comparing the obtained digital signature key with the digital signature of the primary library file, **the primary library file subsequently verifying and selectively loading the plurality of secondary files** by calling the loader program to compare the obtained digital signature key with the digital signature of each of the plurality of secondary files, wherein the computer software installation is a virtual machine installation.

Claim 7 recites

A verification method for a computer software installation, the method comprising the steps of launching a loader program arranged to load files; if a public key is available from an internet site of a virtual machine provider, using the public key as a digital signature key; if a public is not available from the internet site of the virtual machine provider, using a hidden key as the digital signature key; **using the loader program to verify the authenticity of a digital signature incorporated in a primary library file by comparing said digital signature with the digital signature key**; selectively loading the primary library file in dependence upon the successful verification of its digital signature; **using the primary library file and the loader program to verify the authenticity of digital signatures incorporated in each of a plurality of secondary files by comparing them with the digital signature key**;

> and, selectively loading the plurality of secondary files in dependence upon the successful verification of their digital signatures, wherein the computer software installation is a virtual machine installation.

Claim 22 recites

> A system for a computer software installation, comprising: a virtual machine primary library file, the virtual machine primary library file having a digital signature; a loader program that obtains a digital signature key and further loads the virtual machine primary library file; and a plurality of secondary files referenced by the virtual machine primary library file, each of the plurality of secondary files having a digital signature; wherein **the loader program verifies and selectively loads the virtual machine primary library file** by comparing the obtained digital signature key with the digital signature of the virtual machine primary library file, **the virtual machine primary library file subsequently verifying and selectively loading the plurality of secondary files** by calling the loader program to compare the obtained digital signature key with the digital signature of each of the plurality of secondary files, wherein the computer software installation is a virtual machine installation.

Applicant has identified problems with the current art; e.g., a hacker can alter the behavior of the JVM outside the JVM environment so as to undermine the whole Java security model (page 1, line 25, through page 2, line 4) such as by disabling the security code or by inserting destructive routines into the core of the JVM (page 2, lines 13-17). The present invention provides a scheme for verification of the authenticity of a JVM using digital signatures and offers advantages. These advantages include 1) enhanced security of the JVM, 2) greater user confidence in the correct function of Java applications, and 3) improved detection of incorrect or damaged JVM installations (page 9, line 20, through page 10, line 2, of Applicant's specification).

Shear discloses techniques for certifying load modules such as executable computer programs or fragments by a protected or secure processing environment (column 1, lines 25-28). Shear discloses (column 9, lines 42-51):

> FIG. 2 shows how a verifying authority 100 can prevent the problems shown in FIG. 1. In this example, authorized provider 52 submits load modules 54 to verifying authority 100. Verifying authority 100 carefully analyzes the load modules 54 (see 102), testing them to make sure they do

> what they are supposed to do and do not compromise or harm system 50. If a load module 54 passes the tests verifying authority 100 subjects it to, a verifying authority may affix a digital "seal of approval" (see 104) to the load module.

Shear's disclosure of an authorized provider submitting load modules to a verifying authority 100 is in contrast to the claim language in which the loader program verifies the primary library file. For example, claim 1 recites "**the loader program verifies and selectively loads the primary library file**" and "**the primary library file subsequently verifying and selectively loading the plurality of secondary files.**" Shear, in Figure 3, does not show a loader program that verifies and selectively loads the primary library file nor the primary library file subsequently verifying and selectively loading a plurality of files. Although Shear (column 6, lines 5-15) discloses an execution environment protects itself by deciding – based on digital signatures, for example – which load modules or other executables it is will to execute, Shear again does not disclose a loader program that verifies and selectively loads the primary library file nor the primary library file subsequently verifying and selectively loading a plurality of files.

Shear discloses (column 5, lines 10-25):

> A web of trust may stand behind a verifying authority. For example, a verifying authority may be an independent organization that can be trusted by all electronic value chain participants not to collaborate with any particular participant to the disadvantage of other participants. A given load module or other executable may be independently certified by any number of authorized verifying authority participants. If a load module or other executable is signed, for example, by five different verifying authority participants, a user will have (potentially) a higher likelihood of finding one that they trust. General commercial users may insist on several different certifiers, and government users, large corporations, and international trading partners may each have their own unique "web of trust" requirements. This "web of trust" prevents value chain participants from conspiring to defraud other value chain participants.

In contrast to a separate verifying authority as found in Shear, the claimed invention recites the loader program as verifying the primary library file. Furthermore, in the claimed invention the primary library file verifies selected secondary files.

Bodrov shows a dynamic connection between two executable images 100, 200 (Figure 2). Bodrov, in Figure 1, shows an executable image 100 that is a data object and that is dynamically connectable with other executable images, but a loader program that verifies and selectively loads

the primary library file nor the primary library file subsequently verifying and selectively loading a plurality of files. Quoting from Bodrov (column 3, lines 12-24):

> The executable image 100 is a data object that can define by itself or in conjunction with other executable images, one or more software applications. The software applications may include, for example: a word processor, a database, a digital rights management system, a personal finance utility, a graphics tool, an Internet browser, a computer game, a communications program, an authorization program, an electronic wallet, a multi-media renderer or a contract manager. Furthermore, the executable image 100 is dynamically connectable with other executable images. For example, in an embodiment of the invention that is developed for use with the Windows 95, the executable image is a dynamic link library (DLL).

Bodrov does not disclose or suggest "**the loader program verifies and selectively loads the primary library file**" or "**the primary library file subsequently verifying and selectively loading the plurality of secondary files.**"

Claim 1, similar to claims 7 and 22, recites "wherein the loader program is arranged to verify and selectively load the primary library file by comparing the obtained digital signature key with the digital signature of the primary library file" and "selectively loading the plurality of secondary files in dependence upon the successful verification of their digital signatures." Shear discloses a verifying authority that is separate from a loader. Bodrov discloses a dynamic connection between two executable images, but no loader program that both loads and verifies. McManis is limited to the mutual verification by two applications. McManis's verifier is not a "loader program." Just because a software module is able to verify the authenticity of an application does not mean that it will also load such application.

A loader, as defined by http://www.webopedia.com/TERM/l/loader.html, is "an operating system utility that copies programs from a storage device to main memory, where they can be executed. In addition to copying a program into main memory, the loader can also replace virtual addresses with physical addresses. Most loaders are transparent, i.e., you cannot directly execute them, but the operating system uses them when necessary."

McManis discloses (abstract) "The program module verifier responds to procedure calls by verifying the authenticity of any specified program module and by returning a verification confirmation or denial. When the program module verifier fails to verify the authenticity of a

program module, the calling program module throws an exception and aborts its execution."

A loader is not a verifier. McManis does not disclose or suggest a loader.

Applicant asserts that McManis discloses a verifier that is usable by a calling application and a called application, but does not disclose a loader arranged to obtain a digital signature key and further arranged to load the primary library file, as claimed. Since McManis is concerned with preventing use or export of certain cryptographic routines, trade secret functions, and functions protected by contract (column 1, lines 37-63), McManis is not concerned with the basic software but with certain called routines and so it not concerned with the initial loading of a base application, such as a JVM.

Since, none of Shear, Bodrov, or McManis disclose or suggest "**the loader program verifies and selectively loads the primary library file**" or "**the primary library file subsequently verifying and selectively loading the plurality of secondary files,**" claims 1, 2, 5, 7, 8, 11, and 13-23 are allowable over the prior art of record.

Applicant asserts that none of Shear, Bodrov, and McManis discloses or suggests the limitation "wherein after successful verification and selective loading of one of the at least one secondary file, the at least one secondary file is arranged to manage the verification and selective loading of the at least one tertiary file." Thus, it is respectfully submitted that claims 2, 5, 8 and 11 are allowable for this additional reason.

Claims 14 and 15 recite "the virtual machine provider is accessed through an internet site to provide the public key." Shear, in the abstract, Figure 1, or elsewhere, does not disclose this limitation. Thus, claims 14 and 15 are allowable over the prior art for this additional reason.

The Examiner is respectfully requested to reconsider and remove the rejections of the claims 1, 2, 5, 7, 8, 11, and 13-23 under 35 U.S.C. 103(a) based on Shear in view of Bodrov (with or without McManis), and to allow all of the pending claims 1, 2, 5, 7, 8, 11, and 13-23 as now presented for examination. An early notification of the allowability of claims 1, 2, 5, 7, 8, 11, and 13-23 is earnestly solicited.

S.N.: 10/050,083
Art Unit: 2137

Respectfully submitted:

Walter J. Malinowski February 9, 2007

Walter J. Malinowski Date

Reg. No.: 43,423

Customer No.: 29683

HARRINGTON & SMITH, LLP
4 Research Drive
Shelton, CT 06484-6212

Telephone: (203) 925-9400, extension 19
Facsimile: (203) 944-0245
email: wmalinowski@hspatent.com

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, P.O. BOX 1450, Alexandria, VA 22313-1450.

_____ _____

Date Name of Person Making Deposit